Page Flow JSP

Table of contents

1. Introduction

Continuing with the same model used in previous pages of this documentation set, examples will reference this:

implementation flow

Java Page Flow adds a handful of tag libraries to normal JSP usage to assist with the binding of the JSP pages to the controller class. Of primary importance in this topic is the `<netui:xxx>` (../apidocs/taglib/taglib-overview-summary.html#netui) tag library.

Other tag libraries provide additional functionality. The `<netui-data:xxx>` (../apidocs/taglib/taglib-overview-summary.html#netui-data) tag library binds to complex data sets (especially result sets) and renders them as HTML. The `<netui-template:xxx>` (../apidocs/taglib/taglib-overview-summary.html#netui-template) tag library lets you templatize common parts of the presentation layer, so that you can re-use common elements (such as headers, footers, etc.) across the different JSP pages in the web app.

Page Flows also make extensive use of ***databinding expressions*** to bind the JSP pages to data in the controller class. For a detailed explanation of databinding expressions see Databinding: Passing Data Between Controller Classes and JSP Pages (../pageflow/pageflow_databinding.html)

2. Starting a JPF JSP

As with Page Flow Controllers (pageflow_controllers.html) , a certain amount of common boilerplate text is required in each page. The first two lines should set the content-type, the encoding, and import the base netui tag library. The `taglib` binds the netui tags to the `netui` prefix.

```
<%@ page language="java" contentType="text/html;charset=UTF-8"%>
<%@ taglib uri="http://beehive.apache.org/netui/tags-html-1.0"
prefix="netui"%>
```

After the common prolog, the JSP page can be written like most any other JSP page, with some tag substitutions occurring:

original tag	replace with
html	netui:html
body	netui:body
a	netui:anchor

form	netui:form
various form elements	*various netui: elements* (see Handling Forms, below)

In addition to the prolog and general tag substitutions, `<netui:base>` should be present within the `<head>` element. The resulting general form of a JPF-based JSP page is as follows:

```
<%@ page language="java" contentType="text/html;charset=UTF-8"%>
<%@ taglib uri="http://beehive.apache.org/netui/tags-html-1.0"
prefix="netui"%>
<netui:html>
  <head>
    <title>...</title>
    <netui:base/>
  </head>
  <netui:body>
    ..
    ..
    ..
  </netui:body>
</netui:html>
```

3. Simple Linking

Initially, we will examine simple linking through a controller to another JSP page. The `<netui:anchor>` tag replaces the normal `<a>` HTML anchor tag. A plain `<a>` links directly from one URL to another, without providing the controller an opportunity to perform any conditional logic. But the `<netui:anchor>` tag is rendered as HTML and Javascript code, code which causes the link to venture through the controller class.

While it may seem silly to use JPF/netui functionality for simple *constant forward methods*, the advantage is that if a page gets renamed or you wish to change the flow through an application, the destination only needs to be changed once, within the controller. Otherwise, you may have to edit a handful of JSP pages manually changing the URLs inside normal `<a>` tags.

For example, if your application changes and you desire to show a terms-of-service before allowing login, you can simply alter the `login()` controller method to send a user to `terms_of_service.jsp` before further sending him to the actual login screen.

The `<netui:anchor>` tag parallels the `<a>` tag but is able to use an `action` attribute instead of an `href`. Instead of specifying the URL to another page, the name of the method on the controller class is used, without parenthesis.

Instead of using `<a>`

```
<a href="login.jsp">Login!</a>
```

Use `<netui:anchor>`

```
<netui:anchor action="login">Login!</netui:anchor>
```

When the link is displayed on-screen, clicking it will cause control to go through the `Controller`'s `login()` method, which will return the correct forward to select the actual next page to display.

4. Handling Forms

To connect an on-screen form to the underlying controller's form-handling method, the `<netui:form>` container tag, along with specialized tags that replace the normal form elements are used within a JSP page. Similar to how `<netui:anchor>` replaces normal HTML `<a>` tags, the `<netui:form>` tag replaces the typical HTML `<form>` tag. Once again, the tag will render the appropriate page so that submission of the form will venture through the named form-handling method of the controller class.

Instead of using `<form>`

```
<form action="LoginServlet" method="POST">
```

Use `<netui:form>`

```
<netui:form action="processLogin" method="POST">
```

The other tags typically used with a `<form>` also have replacements from the netui tag library.

original tag	replace with	`FormData` datatype
button	netui:button	*optionally changing the controller form processing method*
input type="checkbox"	netui:checkBox	`boolean` or `java.lang.Boolean`
input type="checkbox"	netui:checkBoxGroup	`java.lang.String[]`
input type="checkbox"	netui:checkBoxOption	*see `netui:checkBoxGroup`*
hidden	netui:hidden	`java.lang.String`
input type="radio"	netui:radioButtonGroup	`java.lang.String[]`

input type="radio"	netui:radioButtonOption	*see `netui:radioButtonGroup`*
input type="radio"	netui:select	`java.lang.String[]`
option	netui:option	see `netui:option`
textarea	netui:textArea	`java.lang.String`
input (type="text")	netui:textBox	`java.lang.String`

For the `processLogin(...)` form-processing method, the matching JSP form would be:

```
<netui:form action="processLogin" method="POST">
  <netui:textBox dataSource="actionForm.username" size="20"/>
  <netui:textBox dataSource="actionForm.password" size="20"
password="true"/>
  <netui:button  type="submit" value="Login"/>
</netui:form>
```

When the user submits the form by clicking upon the `Login` button, an instance of the `LoginForm` subclass of `FormData` is created and passed to the `processLogin(LoginForm form)` method of the controller class.

5. Next...

Next, learn about how to compile and package up a complete web-app.

- Building a Page Flow Web-App (pageflow_building.html)